

The Calculus of Failure: Formalizing a Ternary Logic of Structural Wrongness for Safety-Critical Systems

Douglas C. Youvan

doug@youvan.com

www.youvan.ai

May 2, 2026

In the landscape of non-classical logic, three-valued systems have traditionally utilized the third truth value to represent uncertainty, indeterminacy, or a state of being "unknown." While Kleene and Lukasiewicz logics provide robust frameworks for reasoning under partial information, they are fundamentally ill-equipped to handle systems where a third state is not a lack of data, but a definitive presence of corruption. This paper proposes a radical departure from the standard $\{-1, 0, +1\}$ model, introducing a ternary system defined by the set $\{-1, W, +1\}$, where W represents "Wrong." Unlike the neutral zero of traditional ternary logic, W is defined as an absorbing, terminal state. In this paradigm, Wrongness is not a midpoint between True and False, nor is it a superposition that might later resolve into a binary state. Instead, it represents a structural or semantic breach that invalidates the entire logical chain. By treating W as a "poison" value that propagates through operators, we develop a formal calculus designed specifically for environments where the masking of an error is a catastrophic failure. This paper explores the algebraic properties, lattice structures, and practical applications of this "Logic of Error," providing a rigorous foundation for the next generation of fault-tolerant computing and autonomous safety protocols.

Keywords: Ternary Logic, Wrongness Calculus, Error Propagation, Formal Methods, Fault Tolerance, Three-Valued Logic, System Safety, Absorbing Elements, Computational Integrity, Exception Handling, Non-Classical Logic, High-Reliability Systems.

A collaboration with Google Gemini-3.1-Thinking. CC4.0.

Outline

- 1. Introduction** Defining the necessity of a logic that prioritizes error detection over uncertainty management.
- 2. Historical Context and the Limits of Indeterminacy** A review of Kleene and Lukasiewicz logics and why "Unknown" fails in safety-critical hardware contexts.
- 3. Ontology of the Wrong State** Defining W as a structural failure rather than a lack of information or a middle ground between truth values.
- 4. The Algebra of Wrongness: Fundamental Operators** Development of truth tables for NOT, AND, OR, and XOR using the $\{-1, W, +1\}$ set with an emphasis on W -dominance.
- 5. The Absorbing Property and Propagation Dynamics** Mathematical proof of W as an absorbing element in logical expressions and its role in preventing error masking.
- 6. Topological and Lattice Representations** Modeling the system as a flat lattice where W sits outside the linear hierarchy of truth and falsity.
- 7. Implication and Equivalence in Wrong-Valued Logic** Redefining if-then statements and logical equivalence when one or more terms are fundamentally broken.
- 8. Physical Layer Realization and Circuit Design** Strategies for implementing $\{-1, W, +1\}$ in hardware, utilizing high-impedance states or dedicated error-rails.
- 9. Comparative Analysis with Traditional Multi-Valued Logics** A technical breakdown of how "Wrongness Logic" diverges from fuzzy logic and paraconsistent logic.
- 10. Case Study: Aerospace and Autonomous Vehicle Logic Cores** Simulating failure scenarios where W -logic prevents "best-guess" computations in favor of immediate system halts.
- 11. Software Engineering and Compiler Integration** How a ternary logic of W can be mapped to strict exception handling and type systems in modern programming.
- 12. Conclusion and Future Research Directions** Summarizing the benefits of strict error propagation and identifying the path toward W -logic standardizations.
- 13. References** A comprehensive list of foundational texts, academic papers, and technical standards.

Art

1. Introduction

The progression of modern engineering toward increasingly autonomous and safety-critical architectures has exposed a fundamental gap in the classical Boolean logic that underpins our computational systems. While binary logic—comprising the states of True (+1) and False (-1)—has served as the bedrock of digital technology for decades, its inherent simplicity becomes a liability in environments where the integrity of data is as critical as the data itself. In a strictly binary environment, there is no formal mechanism to distinguish between a valid False state and an invalid, corrupted, or structurally compromised state. This lack of distinction forces system designers to rely on external metadata or complex exception-handling routines to manage errors, often leading to silent failures where a corrupted signal is misinterpreted as a valid logical value.

1.1 The Limitations of Binary and Uncertainty-Based Ternary Systems

Classical binary systems operate under the law of the excluded middle, which dictates that any proposition must be either true or false. When applied to real-world hardware, this necessitates that every voltage fluctuation or signal must be mapped to one of two poles. However, hardware is subject to physical degradation, electromagnetic interference, and manufacturing defects. When a logic gate receives a signal that is neither clearly high nor clearly low, or when a memory cell undergoes bit-flipping, a binary system often forces a best guess or defaults to a state that may be factually incorrect. This phenomenon, known as error masking, is the primary driver behind catastrophic failures in autonomous systems where a single bit flip can lead a system to believe a critical sensor is reporting a safe condition when it is actually failing.

Existing three-valued logics, such as those proposed by Kleene or Lukasiewicz, attempted to address this by introducing a third value, often denoted as 0, Unknown, or Indeterminate. In these frameworks, the third value represents a lack of information. These systems are invaluable for database queries and linguistic analysis where a value might exist but is currently hidden. However, they are fundamentally ill-suited for safety-critical hardware. In Kleene logic, for example, the expression (True OR Unknown) evaluates to True. While this is mathematically consistent with the idea that the unknown could be true, it is a dangerous assumption in a flight control system or a medical device. If the Unknown is actually the result of a hardware failure or a sensor breach, treating the overall system state as True or Valid bypasses critical safety checks and masks a potentially fatal error.

1.2 The Conceptual Shift to Structural Wrongness

This paper introduces a formal ternary logic system based on the set $\{-1, W, +1\}$, where W represents Wrong. The shift from Unknown to Wrong is not merely a semantic change but a profound ontological shift in how we treat the third value. In this system, W is not a placeholder

for a future truth or a state of superposition; it is a definitive marker of structural or semantic corruption. Unlike the 0 in traditional ternary systems, which acts as a bridge or a neutral point between extremes, W is defined as an absorbing element. It does not exist on the same spectrum as truth and falsity; rather, it represents the destruction of the spectrum itself.

The defining characteristic of Wrongness is its infectious nature. In a logic chain, the presence of a W value does not signify a state to be resolved; it signifies a state that invalidates all subsequent operations. If a component in a calculation is Wrong, the entire calculation is rendered Wrong. This avoids the pitfalls of error masking found in both binary and Kleene logics. By formalizing W as an absorbing state, we create a logical framework that prioritizes the integrity of the process over the continuation of the computation. In this paradigm, it is better for a system to know it is broken than for it to proceed under a false pretense of validity. This provides a formal mathematical basis for fail-fast architectures, where the logic itself prevents the propagation of corrupt data.

1.3 Scope and Objectives of the Research

The objective of this research is to provide a rigorous mathematical and practical foundation for the $\{-1, W, +1\}$ system. We begin by defining the algebraic properties of Wrong-Valued Logic (WVL), establishing how W behaves under standard operators like NOT, AND, OR, and XOR. We will demonstrate that WVL does not follow the same lattice structures as traditional ternary logics, instead forming a Flat Lattice where W sits outside the comparative hierarchy of truth and falsity. This topological distinction is crucial for understanding why W-logic behaves more like a computer science exception than a mathematical probability.

Beyond the theoretical, this paper explores the physical layer of computation. We will discuss how W can be represented in hardware—moving beyond simple voltage levels to incorporate high-impedance states or dedicated validity rails. We will also examine how this logic can be integrated into high-level programming languages and compilers, providing a type-safe environment where W-propagation replaces or augments traditional exception handling. Finally, through case studies in aerospace and autonomous vehicles, we will demonstrate how the adoption of a formal Logic of Wrongness can prevent the types of failures that currently plague complex, interconnected systems. The ultimate goal of this work is to transition from a logic of "What is the answer?" to a logic of "Is the answer trustworthy?" By embedding the concept of failure directly into the mathematical fabric of the system, we can build architectures that are not just smart, but inherently self-aware regarding their own operational integrity.

2. Historical Context and the Limits of Indeterminacy

The history of formal logic was dominated for over two millennia by the Aristotelian principle of bivalence, which asserts that every proposition must be either true or false. It was not until the early 20th century that mathematicians began to seriously challenge this binary restriction, leading to the development of multi-valued logics (MVLs). While these systems were revolutionary in their attempt to model uncertainty and potentiality, they were designed for philosophical and linguistic applications rather than the unforgiving requirements of digital hardware and real-time safety protocols. Understanding the limitations of these historical systems is essential to appreciating why a dedicated logic of "Wrongness" is necessary for modern engineering.

2.1 The Evolution of Multi-Valued Logics

The formal departure from bivalence began in 1920 with Jan Lukasiewicz, who introduced a third truth value to address the problem of future contingents—propositions about the future that are not yet determined. Lukasiewicz argued that a statement such as "There will be a sea battle tomorrow" is neither true nor false today, but rather "possible" or "indeterminant." This 3-valued system, often denoted as L3, utilized the set $\{0, 1/2, 1\}$, where $1/2$ represented the third state. Lukasiewicz's logic was largely concerned with preserving the potential for truth, treating the middle value as a bridge that might eventually resolve into a binary state.

Following Lukasiewicz, Stephen Cole Kleene developed what is now known as Kleene's Strong Logic of Indeterminacy (K3). Kleene's primary interest was in recursive function theory and the handling of partially defined predicates. In K3, the third value (u) represents a state that is "undecided" or "unknown." This logic was designed to allow a system to continue making progress even when some information was missing. For example, in K3, the truth table for the OR operator is defined such that:

$$1 \text{ OR } u = 1$$

This is mathematically sound if we view (u) as a placeholder for a value that is either 0 or 1, but we simply do not know which one it is yet. Since $(1 \text{ OR } 0 = 1)$ and $(1 \text{ OR } 1 = 1)$, the result is logically guaranteed to be 1 regardless of what (u) eventually becomes.

2.2 The Failure of Unknown in Safety-Critical Hardware Contexts

While the "Unknown" placeholder logic is highly effective for database queries or semantic reasoning, it introduces a dangerous level of optimism into hardware and safety-critical software. In these domains, the third state is rarely a "future contingent" or an "undecided" variable; it is more often the result of a physical or logical failure. If we apply Kleene's K3 logic to a sensor array in an autonomous vehicle, the limitations become immediately apparent.

Consider a scenario where a primary collision-avoidance sensor returns a "True" (+1) signal indicating an obstacle, while a secondary verification sensor returns an "Unknown" (u) signal due to a hardware malfunction or a severed wire. In a K3-based system, the expression (True OR Unknown) simplifies to "True." This effectively masks the fact that the secondary sensor is failing. The system proceeds as if it has a valid, redundant confirmation, when in reality, the integrity of the sensing platform has been compromised. The logic "cleans up" the error by assuming it doesn't matter, which is an unacceptable behavior in high-stakes environments.

2.3 The Semantic Gap: Uncertainty vs. Corruption

The fundamental flaw in historical 3-valued logics, when applied to engineering, is the conflation of uncertainty with corruption. Uncertainty (the "Unknown") implies a lack of data that may be filled later. Corruption (the "Wrong") implies that the data path itself is no longer trustworthy. Standard ternary systems treat the third value as a "weak" state that can be overridden by "strong" binary states. In contrast, a system designed for safety must treat the third value as a "strong" state that overrides binary states.

The set $\{-1, 0, +1\}$ in traditional ternary systems assumes a linear continuum where 0 is a midpoint. This implies that 0 is "less true" than +1 but "more true" than -1. In a safety-critical context, a hardware failure is not a midpoint; it is an orthogonal event. It is a departure from the domain of truth and falsity entirely. Historical logics fail to capture this orthogonality because they remain bound to the idea of a truth-value hierarchy.

2.4 Toward a Logic of Structural Failure

The transition from Kleene's "Unknown" to our proposed "Wrong" (W) requires a total re-evaluation of operator behavior. We must move away from the "optimistic" resolution of operators (where valid data can mask invalid data) toward a "pessimistic" or "strict" propagation model. In such a model, the existence of a single "Wrong" value in a logic chain serves as a signal that the entire chain is suspect. This requires a formal calculus where W is an absorbing element, ensuring that no amount of valid True or False data can bury a structural failure. By looking back at the history of Lukasiewicz and Kleene, we see that while they successfully broke the binary mold, they did so to solve problems of "what we don't know," leaving the problem of "what we have broken" entirely unaddressed. This paper's introduction of the W state is the necessary evolution to bridge that gap.

3. Ontology of the Wrong State

The introduction of W into a ternary logic system requires a rigorous ontological definition to prevent it from being treated as a synonym for "Undefined" or "Null." In the $\{-1, W, +1\}$ system,

the ontology of W is grounded in the concept of structural failure. To understand this, one must move away from the traditional view of truth as a linear gradient. In standard binary or fuzzy logics, values exist on a spectrum where one end is the total absence of truth and the other is its total presence. W does not exist on this spectrum. It is not "partially true" or "mostly false." It represents a departure from the logic itself—a state where the conditions required for a logical proposition to be evaluated have been violated.

3.1 The Departure from the Excluded Middle and the Included Middle

Classical logic relies on the Law of the Excluded Middle, which posits that for any proposition P, either P or not-P must be true. Standard ternary logics introduce an "included middle" (0), suggesting a transition point between the two poles. The ontology of W rejects both. W is not a middle ground; it is an "externalized" state. If we visualize a binary logic as a line connecting two points, and a standard ternary logic as a line with a midpoint, the W state is a point existing off that line entirely. It signifies that the line itself—the medium of logical transmission—has been severed.

This distinction is vital for high-reliability systems. When a sensor fails, it is not providing a "zero" signal in the sense of a neutral value. It is providing a value that cannot be mapped to the intended domain. By defining W as ontologically distinct from -1 and +1, we acknowledge that the failure of the observer or the instrument is a different category of reality than the truth or falsity of the observation itself.

3.2 Wrongness as a Terminal Semantic Breach

In this framework, W is defined by its role as a terminal state. Unlike an "Unknown" value which represents a temporary lack of knowledge that might be resolved, a "Wrong" value represents a corruption that cannot be repaired through further logical operations. If a calculation results in W, it indicates that the semantic meaning of the operation has been lost. For instance, in an arithmetic operation within this logic, if we attempt to divide by a value that is structurally corrupted, the result is not an "undefined" limit but a "Wrong" state that alerts the system to a total loss of computational integrity.

This terminal nature makes W a "poison" or "absorbing" value. The ontology here is one of strict containment. Once a "Wrongness" enters the system, it is treated as a contagion. This reflects the reality of complex software where a single null pointer or buffer overflow doesn't just make a specific variable "unknown"—it makes the entire state of the program suspect. By formalizing this suspicion into a truth value, we allow the logic to reason about its own invalidity.

3.3 The Functional Difference Between False and Wrong

A common point of confusion in multi-valued systems is the distinction between a logical False (-1) and an Error state (W). In the {-1, W, +1} system, -1 is a valid, healthy result. It indicates that a condition was checked and was found to be not present. W, however, indicates that the check itself was invalid.

Consider a safety circuit monitoring a pressure valve. A result of -1 means "the pressure is not too high." This is a successful, actionable piece of information. A result of W means "the pressure sensor has short-circuited." This is not a state of "not high" or "high"—it is a state of "we no longer know the pressure, and our mechanism for knowing it is compromised." By separating these two concepts ontologically, we ensure that the system does not confuse a "failed sensor" with a "safe condition." This clarity is the primary advantage of W-logic over binary systems that might default to a safe-state upon failure, thereby masking a critical hardware defect.

3.4 W as a Metadata Primitive

Ultimately, the ontology of W can be seen as the integration of metadata into the primitive data type. In traditional architectures, the status of a signal (whether it is valid or invalid) is often carried on a separate bus or stored in a separate status register. This separation creates a "split-brain" problem where the logic can process data while being unaware that its validity bit has been flipped. In a {-1, W, +1} system, the validity is the data. There is no way to access the value without also acknowledging its integrity. This unification of value and validity is what gives the "Wrong" state its ontological power, transforming it from a mere error code into a fundamental constituent of the logical universe.

4. The Algebra of Wrongness: Fundamental Operators

To formalize the {-1, W, +1} system, we must establish a set of algebraic rules that govern how these values interact. While binary logic relies on Boolean algebra and traditional ternary systems often use Kleene or Lukasiewicz algebras, Wrong-Valued Logic (WVL) requires a unique operator set where the W state acts as a dominant, absorbing element. This mathematical behavior ensures that any structural failure is propagated through the logic chain rather than being suppressed by valid binary values. In this section, we define the truth tables and algebraic properties of the four fundamental operators: NOT, AND, OR, and XOR.

4.1 The Unary Operator: NOT

The NOT operator (negation) in WVL is designed to reflect the principle that an invalid state cannot be inverted into a valid one. In classical logic, NOT (+1) equals -1, and NOT (-1) equals +1.

In a "Wrong" logic system, however, the negation of a structural failure remains a structural failure. There is no "inverse" to a broken state. If the input is fundamentally compromised, its logical opposite is equally compromised because the basis for the comparison has been destroyed.

The truth table for NOT (A) is defined as follows:

A | NOT A

+1| -1

-1| +1

W | W

This behavior distinguishes WVL from systems where the third value represents a midpoint. In some fuzzy logics, the negation of the midpoint is the midpoint itself because it represents a 0.5 probability. In WVL, the result is W because W signifies a lack of semantic standing. If a statement is Wrong, saying "it is not the case that this statement is Wrong" does not yield a True or False result; it merely confirms that the logic core cannot process the request.

4.2 The Conjunction Operator: AND

The AND operator (conjunction) in WVL follows a strict propagation model. In binary logic, A AND B is true only if both are true. In Kleene's K3 logic, if one side is False and the other is Unknown, the result is False because the "Unknown" could never make the "False" side True. WVL rejects this optimism. If any part of a conjunction is Wrong, the entire expression is invalidated. The logic assumes that if one input is structurally compromised, the total operational context of the AND gate is suspect.

The truth table for A AND B is:

A | B | A AND B

+1 | +1 | +1

+1 | -1 | -1

+1 | W | W

-1 | +1 | -1

-1 | -1 | -1

-1 | W | W

W | +1 | W

W | -1 | W

W | W | W

In this model, W functions as a "poison" value. Even if one input is a definitive False (-1), which in binary logic would be enough to determine the result, the presence of a W forces the output to W. This is a critical safety feature: it prevents a system from concluding that a condition is "False" when it actually has no reliable data to make that determination.

4.3 The Disjunction Operator: OR

The OR operator (disjunction) is where WVL diverges most sharply from historical ternary systems. In Kleene logic, (True OR Unknown) evaluates to True. In WVL, (True OR Wrong) evaluates to Wrong. This is the "Strict Error" principle. While it may seem counterintuitive to invalidate a True signal just because another signal is Wrong, this behavior is essential for high-integrity computing.

The truth table for A OR B is:

A	B	A OR B
---	---	--------

+1	+1	+1
----	----	----

+1	-1	+1
----	----	----

+1	W	W
----	---	---

-1	+1	+1
----	----	----

-1	-1	-1
----	----	----

-1	W	W
----	---	---

W	+1	W
---	----	---

W	-1	W
---	----	---

W	W	W
---	---	---

By defining OR in this manner, we ensure that a "Success" state (+1) cannot mask a "Failure" state (W). In an aerospace context, if an engine sensor says "All Clear" (+1) but the fire detection logic returns "Wrong" (W) due to a short circuit, the resulting system state must be W. Masking the W with the +1 would lead the pilot to believe the system is healthy when it is actually operating with a blinded safety component.

4.4 The Exclusive Disjunction: XOR

The XOR operator in WVL is used to detect differences between states. In binary logic, XOR returns True if the inputs are different. In WVL, XOR maintains the absorbing property of W. If either side of the comparison is Wrong, the comparison itself is impossible to perform reliably.

The truth table for A XOR B is:

A | B | A XOR B

A | B | A XOR B

+1 | +1 | -1

+1 | -1 | +1

+1 | W | W

-1 | +1 | +1

-1 | -1 | -1

-1 | W | W

W | +1 | W

W | -1 | W

W | W | W

This definition of XOR is particularly useful for redundancy checking. In a Dual Modular Redundancy (DMR) system, two processors perform the same task and their results are XORed. If the results are different, an error is flagged. In a WVL-based system, if one processor returns a valid result and the other returns W, the XOR output is W, immediately signaling a structural failure rather than a simple data mismatch.

4.5 Algebraic Identities and the Dominance of W

The algebraic structure of WVL can be summarized by several key laws. Notably, W functions as the identity for certain failure-analysis operations but, more importantly, it serves as the Null Element for the entire system under AND and OR operations.

1. Commutativity: $A \text{ AND } B = B \text{ AND } A$ $A \text{ OR } B = B \text{ OR } A$
2. Associativity: $(A \text{ AND } B) \text{ AND } C = A \text{ AND } (B \text{ AND } C)$ $(A \text{ OR } B) \text{ OR } C = A \text{ OR } (B \text{ OR } C)$
3. Absorption by W: $A \text{ AND } W = W$ $A \text{ OR } W = W$

The third law, Absorption by W, is the defining characteristic of this algebra. In standard Boolean algebra, the null element for AND is False (0), and the identity for OR is False (0). In WVL, W overrides both. This creates a "Failure-Priority" logic. The existence of W represents a boundary condition that the logic cannot cross. Consequently, the distribution laws still hold, but they are always subject to the presence of W. For example, $A \text{ AND } (B \text{ OR } C)$ will equal W if any of the

variables A, B, or C are W. This algebraic rigidity is what provides the mathematical guarantees required for formal verification in safety-critical systems.

5. The Absorbing Property and Propagation Dynamics

The defining mathematical characteristic of Wrong-Valued Logic (WVL) is the elevation of the W state to an absorbing element across all primary binary operators. In classical algebraic structures, an absorbing element (also known as a null element) is a member of a set that, when combined with any other member of the set via a binary operation, results in the absorbing element itself. In the $\{-1, W, +1\}$ system, W acts as this dominant force, ensuring that the structural integrity of a logical expression is never compromised by the "optimistic" resolution of binary values. This section provides the formal proof of these properties and examines the dynamics of how W propagates through complex logical networks.

5.1 Mathematical Proof of W as an Absorbing Element

To establish W as an absorbing element, we define the set $S = \{-1, W, +1\}$ and consider the binary operations of conjunction (AND) and disjunction (OR). In standard Boolean algebra, the absorbing element for AND is False (-1), as $(A \text{ AND } -1 = -1)$ for all A. However, in WVL, the priority shifts from truth-determination to integrity-determination.

For the operation AND, we state the absorption law as: For all x in S, $x \text{ AND } W = W$. Proof by exhaustion: $(+1) \text{ AND } W = W$ $(-1) \text{ AND } W = W$ $(W) \text{ AND } W = W$ Since all possible values of x combined with W yield W, W is the absorbing element for the AND operation.

For the operation OR, we state the absorption law as: For all x in S, $x \text{ OR } W = W$. Proof by exhaustion: $(+1) \text{ OR } W = W$ $(-1) \text{ OR } W = W$ $(W) \text{ OR } W = W$ This marks a significant departure from Kleene logic (K3), where $(+1) \text{ OR Unknown} = (+1)$. In WVL, the OR operator is "strict" or "pessimistic." The mathematical justification for this is that an OR gate cannot reliably determine a "True" outcome if one of its inputs is structurally compromised. The "Wrongness" of one branch invalidates the entire disjunction.

5.2 Propagation Dynamics in Deep Logic Chains

In complex computational structures, logical operations are rarely isolated. They form deep chains where the output of one gate serves as the input for the next. The absorbing property of W creates a specific propagation dynamic known as "Poisoning." When a W value is generated at any point in a logic tree—whether due to a hardware fault, a buffer overflow, or a sensor failure—it begins to move through the network, "poisoning" every subsequent gate it touches.

Consider a nested logic expression: $\text{Result} = ((A \text{ AND } B) \text{ OR } (C \text{ AND } D)) \text{ AND } E$

If variable C is found to be Wrong ($C = W$), the propagation follows this sequence:

1. $(C \text{ AND } D)$ becomes $(W \text{ AND } D)$, which equals W.
2. The expression simplifies to $((A \text{ AND } B) \text{ OR } W) \text{ AND } E$.
3. The OR operation $((A \text{ AND } B) \text{ OR } W)$ evaluates to W.
4. The final expression $(W \text{ AND } E)$ evaluates to W.

Regardless of the values of A, B, D, or E, the result is W. This dynamic ensures that an error is never "lost" in the logic. In traditional systems, if A and B were both True, the $(A \text{ AND } B)$ branch might satisfy the OR condition, allowing the system to proceed with a valid result while being entirely unaware that the C/D branch was failing. WVL propagation dynamics eliminate this blind spot by ensuring that the W signal always reaches the final output of the logic chain.

5.3 Preventing Error Masking

Error masking is a phenomenon in digital systems where a fault occurs but does not affect the output of a circuit, leading to "latent" failures that can strike catastrophically later. Binary logic is highly susceptible to masking. For example, in a 2-input OR gate, if one input is stuck-at-high (+1), any failure on the other input is masked because the output will always be +1.

In WVL, the "Wrong" state is designed specifically to prevent masking. Because W is an absorbing element, it cannot be "overwritten" by a +1 or -1. This transforms the logic from a system that merely calculates values into a system that simultaneously calculates its own validity. The propagation of W is essentially a continuous, real-time diagnostic signal that travels alongside the data. This eliminates the need for separate parity bits or checksums at every gate, as the data format itself carries the status of its own structural health.

5.4 Computational Complexity and Efficiency of Propagation

While it may seem that a "poisonous" logic would be overly sensitive, it actually provides significant benefits in terms of computational efficiency for safety-critical monitors. In a standard system, a supervisor must check every component's status bit individually. In a WVL system, the supervisor only needs to check the final output. If the output is not W, the system is mathematically guaranteed that every single component involved in that specific calculation was healthy (i.e., not W).

The propagation of W can be viewed as an "eager" error handling mechanism. Instead of waiting for a calculation to finish and then checking for errors, the logic chain effectively self-terminates. In hardware implementations, this can be used to trigger immediate gate-level

shutdowns or to divert processing power away from corrupted sub-systems. The absorbing property thus serves a dual purpose: it provides a rigorous mathematical guarantee against error masking and serves as a highly efficient signaling mechanism for system-wide fault tolerance. This behavior forms the basis for the "Fail-Fast" architecture explored in later sections, where the logic itself acts as the primary firewall against the propagation of corrupted data.

6. Topological and Lattice Representations

The mathematical structure of a logic system is best understood through its lattice representation, which provides a visual and algebraic map of how truth values relate to one another. In classical Boolean logic, the truth values form a simple two-element lattice where -1 is less than $+1$. In traditional ternary systems, this is expanded into a linear chain. However, the $\{-1, W, +1\}$ system breaks this linear progression, requiring a topological shift to a flat lattice. This section explores the structural properties of this lattice and the implications for how information and error are categorized within the system.

6.1 The Failure of Linear Ordering

In most multi-valued logics, the third value is treated as a midpoint on a scale of certainty. This creates a linear order expressed as $-1 < 0 < +1$. In such a system, the values are comparable; 0 is "more true" than -1 but "less true" than $+1$. This ordering is crucial for fuzzy logic and probabilistic reasoning where we want to measure degrees of truth.

The ontology of WVL rejects this hierarchy entirely. Because W represents a structural failure rather than a degree of truth, it cannot be meaningfully compared to -1 or $+1$. It is not "greater than" a False state, nor is it "less than" a True state. To say that a broken sensor is better or worse than a sensor reporting a safe condition is a category error. Consequently, the $\{-1, W, +1\}$ set is partially ordered rather than totally ordered. In this system, $+1$ and -1 are incomparable to W .

6.2 Defining the Flat Lattice Structure

The topology of WVL is most accurately represented as a flat lattice. In a flat lattice, there is a top element (representing the maximum or "All") and a bottom element (representing the minimum or "Empty"), but the primary values of the logic exist on a single horizontal plane between them. In our specific implementation of W-logic, we define the structure based on operational integrity.

In this diagram, $+1$ and -1 are at the same hierarchical level, representing valid logical states. W , however, occupies a distinct topological position. While it is often represented on the same

horizontal plane to denote its incomparability, in safety-critical modeling, it is frequently placed as a "sink" or an orthogonal point. This visualization reinforces the idea that W is an exit point from the logic rather than a step on the way to a conclusion.

6.3 Join and Meet Operations in WVL

In lattice theory, the two fundamental operations are the Join (least upper bound) and the Meet (greatest lower bound). For a lattice $(S, \leq)$, the Join of two elements a and b is the smallest element that is greater than or equal to both. The Meet is the largest element that is less than or equal to both.

In WVL, because W is incomparable to the binary states, the Join and Meet operations must be redefined to account for the "Wrong" state. Let $S = \{-1, W, +1\}$. If we define a partial order where $x \leq y$ only if $x = y$, then the Meet (AND) and Join (OR) for any binary state and W always results in an undefined or "Top" error state. In our specific algebra defined in Section 4, we see this reflected:

$$A \text{ (Meet) } W = W \quad A \text{ (Join) } W = W$$

This shows that W acts as both the Meet and Join whenever it is present. This is a rare property in lattice theory and highlights the "absorbing" nature of the system. Mathematically, this suggests that the $\{-1, W, +1\}$ system is not just a lattice but a "strict" lattice where the presence of the error state collapses the distinction between upper and lower bounds.

6.4 Information Order vs. Truth Order

A critical distinction in the topology of WVL is the difference between truth order and information order. Standard ternary logics often use information order (also known as the Scott order) where 0 is at the bottom because it represents "no information," while -1 and $+1$ are at the top because they represent "complete information."

WVL flips this concept. In our system, W does not represent a lack of information; it represents the presence of misinformation or structural corruption. Therefore, W cannot be at the bottom of an information lattice. If we were to construct an information lattice for WVL, W would be an "isolated" or "divergent" node. It represents a state where the information density is high (we know exactly what is wrong) but the logical utility is zero. This topological isolation is what prevents W from being masked or optimized away by compilers and logic synthesizers.

6.5 Geometric Interpretation of Operator Dynamics

By viewing WVL through a topological lens, we can interpret logical operations as movements across the lattice. Binary operations (AND/OR) between -1 and $+1$ result in movements between

the two valid poles. However, any operation involving W acts as a "gravity well" that pulls the state of the system off the valid plane and into the W-sink.

This geometric interpretation is useful for formal verification. When modeling a system's state space, any path that enters the W-sink is immediately flagged as a failure state. Because there are no paths leading from W back to the valid plane (+1 or -1) within the fundamental algebra, the W-sink is a "trap state." This topological guarantee is what allows engineers to prove that once a structural failure occurs, the system will not accidentally return to an "apparently healthy" state without an external reset or intervention. The lattice is not just a map of truth; it is a map of the system's safety boundaries.

7. Implication and Equivalence in Wrong-Valued Logic

In any formal logic, the mechanisms of implication and equivalence determine how truth is transferred from one proposition to another. In classical Boolean logic, the material implication (A implies B) is defined by its truth table such that the only way to produce a false result is for a true premise to lead to a false conclusion. However, in the $\{-1, W, +1\}$ system, the introduction of the "Wrong" state necessitates a complete re-evaluation of these relationships. Since W is an absorbing state representing structural failure, it cannot participate in the vacuous truths or logical identities common to binary systems. This section defines the behavior of implication and equivalence when the integrity of the proposition itself is in question.

7.1 The Material Implication Operator

In Wrong-Valued Logic (WVL), we define the implication operator ($A \rightarrow B$) through the lens of strict error propagation. In classical systems, the implication is often defined as $(\text{NOT } A \text{ OR } B)$. Following the strict OR and NOT operators established in Section 4, the WVL implication must yield W if either the antecedent (A) or the consequent (B) is W. This creates a system where a structural failure at any point in an "if-then" statement renders the entire inference invalid.

The truth table for $(A \rightarrow B)$ is as follows:

A	B	$A \rightarrow B$
---	---	-------------------

+1	+1	+1
----	----	----

+1	-1	-1
----	----	----

+1	W	W
----	---	---

-1	+1	+1
----	----	----

-1	-1	+1
----	----	----

-1	W	W
----	---	---

W	+1	W
---	----	---

W	-1	W
---	----	---

W	W	W
---	---	---

This table highlights a critical departure from classical logic: the rejection of vacuous truth for Wrong antecedents. In binary logic, if A is false, the statement $(A \rightarrow B)$ is always true regardless of B. In WVL, if A is -1 (False), the implication still functions classically, yielding +1. However, if A is W (Wrong), the implication does not become "True" (+1); it becomes W. This is because a "Wrong" premise is not merely a false statement—it is a broken signal. One cannot draw a valid logical inference from a broken signal, even if the consequent appears to be true. This prevents the system from validating logic chains that are built upon corrupted hardware or semantic errors.

7.2 Logical Equivalence and the Identity Crisis

Logical equivalence $(A \Leftrightarrow B)$ is typically defined as $(A \rightarrow B) \text{ AND } (B \rightarrow A)$. It represents the idea that two propositions share the same truth value. In WVL, equivalence is subject to the same "poisoning" effect as other operators. If we compare two valid states, the operator behaves classically: $(+1 \Leftrightarrow +1)$ is +1, and $(-1 \Leftrightarrow -1)$ is +1. However, the behavior of W in an equivalence relation is the most distinct feature of this logic.

Specifically, the expression $(W \Leftrightarrow W)$ evaluates to W.

This is a profound shift. In many multi-valued logics, an unknown value is often considered equal to another unknown value. In WVL, because W represents a structural failure, the comparison itself is a failure. If two sensors are both "Wrong," we cannot say they are "Equivalent" in a way that yields a valid "True" (+1) signal. To do so would allow a system to bypass safety checks simply because two errors happened to match. By defining $(W \Leftrightarrow W)$ as

W, we ensure that equivalence remains a measure of valid data identity, not a measure of shared corruption.

7.3 The Dissolution of Classical Tautologies

A tautology is a formula that is true under every possible interpretation of its variables. In binary logic, the Law of the Excluded Middle ($A \text{ OR } \text{NOT } A$) is a fundamental tautology. In WVL, this law no longer holds. If A is W , then $(W \text{ OR } \text{NOT } W)$ simplifies to $(W \text{ OR } W)$, which results in W .

Consequently, WVL is not a "truth-preserving" logic in the classical sense; it is an "integrity-preserving" logic. Most classical tautologies fail when a variable is replaced by W . This is an intentional design choice. In a safety-critical system, we do not want a "Universal Truth" to override a local structural failure. For instance, in a binary software system, a tautology like $(x == x)$ might evaluate to true even if (x) is a corrupted pointer, potentially leading to a crash later in the execution. In a WVL-aware compiler, if (x) is W , the comparison $(x == x)$ results in W , triggering an immediate error-handling routine rather than proceeding with a false sense of security.

7.4 If-Then-Else Constructs and Branching

The implications of W -logic extend to the control flow of software. In traditional programming, an "if" statement expects a boolean result. If the condition is anything else, the system usually defaults to the "else" branch or throws a runtime exception. In a system utilizing WVL, a branching statement based on a "Wrong" condition cannot proceed to either the "then" or the "else" branch.

Because $(A \rightarrow B)$ results in W when A is W , the branch itself becomes W . This necessitates a third path in control flow—the "Failure Branch." This allows for a more granular approach to error handling than the "try-catch" blocks found in modern languages. Instead of jumping to a distant exception handler, the logic naturally flows into a state of "Wrongness" that can be handled locally or propagated upward. This ensures that the decision-making process of an autonomous system is always grounded in valid data, as the logic itself lacks the "vocabulary" to make a binary choice when faced with structural corruption.

7.5 Functional Completeness and the Power of W

In logic, a set of operators is functionally complete if every possible truth function can be expressed using only those operators. While WVL limits the "reach" of certain operators by making W an absorbing element, it remains functionally complete regarding the valid states (-1 and $+1$). The presence of W acts as a boundary or a "guard" around the functional core.

The algebra of implication in WVL ensures that the logic is "closed" under error. There is no combination of AND, OR, NOT, or IF-THEN that can take a W input and produce a $+1$ or -1 .

output. This closure is the mathematical proof of the system's safety. It guarantees that once a component enters the "Wrong" state, no subsequent logical manipulation—no matter how complex or convoluted—can "cleanse" that error and turn it back into a valid signal. This property is what makes WVL a robust foundation for formal verification, as it allows engineers to prove the non-masking of errors through the entire depth of a logic circuit.

8. Physical Layer Realization and Circuit Design

Translating the abstract algebraic properties of Wrong-Valued Logic (WVL) into physical hardware requires a departure from standard CMOS binary logic. In a binary system, information is typically represented by two voltage levels, V-high and V-low. To implement the set $\{-1, W, +1\}$, we must define three distinct physical states and, crucially, design gates that physically enforce the "absorbing" property of W. This section explores several hardware strategies, ranging from multi-level voltage windows to dual-rail asynchronous signaling, to manifest "Wrongness" as a tangible electrical state.

8.1 Multi-Threshold Voltage Mapping

The most direct way to represent a ternary system is through three distinct voltage levels on a single wire. In a traditional ternary computer (such as the Soviet Setun), these were often positive voltage, zero voltage, and negative voltage. In a WVL-optimized circuit, we might map the states as follows:

1. +5V (or V_{dd}) represents +1 (True).
2. -5V (or V_{ss}) represents -1 (False).
3. 0V (Ground) represents W (Wrong).

While this mapping appears intuitive, it presents a significant engineering challenge: distinguishing between a "Wrong" state and a "Power-Off" state. If a wire is severed, it naturally floats toward 0V. While this effectively maps a physical break to the W state—consistent with our ontology of structural failure—it requires the receiving gates to be "active-low" for the W state. To enhance reliability, a windowed approach can be used where any voltage between -1V and +1V is interpreted as W by a Schmitt trigger or a specialized comparator. This ensures that signal degradation or electromagnetic noise that pulls a signal toward the middle will be treated as a structural failure rather than being rounded to the nearest binary pole.

8.2 High Impedance (Hi-Z) and Open-Collector Logic

A more sophisticated realization of W is the use of high-impedance (Hi-Z) states. In standard tri-state logic, Hi-Z is used to "disconnect" a component from a shared bus. In WVL, we can

repurpose this state to represent W. Under this paradigm, a "Healthy" gate provides a low-impedance path to either the +1 or -1 voltage rail. A "Wrong" gate enters a high-impedance state, essentially "floating" its output.

This has a unique advantage for W-propagation. By using "weak pull-up" or "weak pull-down" resistors on every logic line that are tuned to a specific "W-reference" voltage, any gate that fails or enters a Hi-Z state will automatically pull the entire line to the W-level. This creates a physical version of the absorbing property. If a driver in a logic chain fails to provide a strong +1 or -1 signal, the line defaults to the W-state, and every subsequent gate—designed to detect the absence of a strong binary signal—will propagate that W-state forward.

8.3 Dual-Rail Encoding for Fault Detection

For maximum safety-critical reliability, the most robust implementation of WVL is through Dual-Rail Encoding. In this architecture, every logical signal is carried by two physical wires (Wire-A and Wire-B). The states are mapped as follows:

- (Wire-A=High, Wire-B=Low) = +1
- (Wire-A=Low, Wire-B=High) = -1
- (Wire-A=Low, Wire-B=Low) = W (Absence of signal)
- (Wire-A=High, Wire-B=High) = W (Illegal state)

In this system, W is represented by any "unbalanced" state. This is a form of self-timed or asynchronous logic. The primary benefit of this realization is that it catches both passive failures (e.g., a wire being cut results in Low-Low) and active failures (e.g., a short circuit to the power rail results in High-High). A WVL-logic gate in a dual-rail system consists of combinational logic that checks the validity of both input rails. If the inputs to an AND gate are (High-Low) and (Low-High), it outputs (Low-High). But if any input pair is (Low-Low) or (High-High), the output rail is immediately forced to a "Wrong" state. This provides a hardware-level guarantee of the "poisoning" effect described in earlier sections.

8.4 Transistor-Level Implementation of the W-AND Gate

To illustrate the gate design, consider a WVL AND gate using a windowed voltage approach. Standard CMOS AND gates use a series/parallel arrangement of PMOS and NMOS transistors. A WVL gate requires additional "integrity-sensing" transistors.

In a simplified ASCII representation of a WVL AND gate path:

```
Input A ---[Sense Circuit]---+ | Input B ---[Sense Circuit]---+---[Decision Logic]---> Output | W-
Reference -----+
```

The "Sense Circuit" monitors the input voltage. If the voltage is within the valid (+1 or -1) windows, the "Decision Logic" performs a standard binary AND. However, if the Sense Circuit detects that either Input A or Input B is within the W-window (e.g., near 0V), it triggers a bypass switch that connects the Output directly to the W-Reference. This "crowbar" effect ensures that the W-state overrides any potential binary output. This physical dominance is the hardware manifestation of W-absorption. Unlike binary gates where a "0" on one input of an OR gate is ignored, a "W" on a WVL OR gate physically shunts the output to the error state.

8.5 Propagation Delays and Signal Integrity

A unique challenge in WVL hardware is managing the propagation delay of the W-signal. In many safety-critical systems, it is desirable for the W-signal to travel faster than the data it replaces. By using dedicated "validity rails" or high-speed comparators, a "Wrongness" event can be propagated through the system at the speed of the electrical transient, effectively "clearing the path" or resetting downstream registers before the corrupted data signal can be latched.

This realization leads to the concept of "Active-Wrong" hardware. Instead of W being a passive state (like a lack of signal), it can be an active high-frequency pulse that is distinct from DC levels. If a gate detects an internal failure (such as a parity error in its own internal logic), it emits this W-pulse. Subsequent gates are designed to be "quenched" by this pulse, stopping all data processing and entering a safe-state. This ensures that the logical "Wrongness" is not just a value in a table, but a physical override that protects the system from the unpredictability of corrupted silicon.

8.6 Resilience to Signal Degradation

Finally, WVL hardware provides superior resilience to signal degradation compared to traditional ternary or fuzzy systems. Because W is an absorbing state that exists outside the truth hierarchy, the system can use very wide "guard bands" for its binary states. For example, any voltage from +3V to +5V could be +1, while any voltage from -3V to -5V is -1. The entire middle range (from -2.9V to +2.9V) is W. This massive "Wrongness" window ensures that even significant signal attenuation or crosstalk does not result in a bit-flip (where a -1 is mistaken for a +1). Instead, the noise simply pushes the signal into the W-domain, where it is caught and handled by the logic's inherent error-propagation rules. This makes WVL hardware exceptionally well-suited for high-interference environments like satellite communications or industrial robotics.

9. Comparative Analysis with Traditional Multi-Valued Logics

To fully appreciate the utility of Wrong-Valued Logic (WVL), it must be situated within the broader landscape of Non-Classical Logics. While the transition from two-valued to three-valued systems is a well-trodden path in mathematical logic, the intent behind the third value varies wildly across different frameworks. Most traditional multi-valued logics (MVLs) were designed to handle shades of gray—uncertainty, partiality, or paradox. WVL, conversely, is a logic of black, white, and a "broken switch." This section compares WVL against the most prominent MVLs to highlight its unique behavior regarding error propagation and system integrity.

9.1 WVL vs. Kleene's K3 and Łukasiewicz's L3

The most significant comparison lies with Kleene's Strong Three-Valued Logic (K3) and Łukasiewicz's L3. Both systems utilize a third value, typically denoted as 0 or 1/2, to represent "Unknown" or "Indeterminate." The fundamental difference is found in their approach to truth-functionality. K3 is essentially "optimistic." It operates on the principle that if a result can be determined despite an unknown variable, it should be. For example, in K3, (True OR Unknown) results in True. The reasoning is that regardless of whether the unknown eventually resolves to True or False, the OR condition is satisfied.

WVL rejects this optimism entirely. In a safety-critical context, (True OR Wrong) results in Wrong. The WVL perspective is that the presence of a "Wrong" state indicates a structural failure in the logic's input. Even if a "True" signal is present on one branch, the fact that the other branch is "Wrong" suggests that the evaluator (the gate or the software) cannot be certain of the overall system state. While K3 seeks to maximize the reach of truth, WVL seeks to maximize the detection of failure. In K3, the third value is a "weak" state that can be overridden; in WVL, the third value is a "strong" state that overrides all others.

9.2 Contrast with Fuzzy Logic

Fuzzy logic is often conflated with ternary logic because both deal with values beyond a binary pair. However, their ontologies are diametrically opposed. Fuzzy logic is a logic of "degree." It maps truth to the real interval $[0, 1]$, where a value like 0.7 represents "mostly true." It is designed to handle vagueness in natural language or sensor noise—situations where a precise boundary is difficult to define.

WVL is not a logic of degree. There is no such thing as "mostly Wrong" or "slightly Wrong." In WVL, the values -1 and +1 remain crisp and absolute. The W state is a categorical departure from the truth-spectrum. If fuzzy logic is a dimmer switch, WVL is a circuit breaker. While fuzzy logic attempts to smooth out the transition between states to provide a nuanced output, WVL introduces a hard-stop that prevents nuanced (but potentially corrupted) data from being used

in critical decisions. WVL is concerned with "integrity," whereas fuzzy logic is concerned with "precision."

9.3 WVL vs. Paraconsistent Logics

Paraconsistent logics, such as Graham Priest's Logic of Paradox (LP), are designed to handle contradictions without resulting in "explosive" consequences. In classical logic, a single contradiction allows you to prove anything (ex falso quodlibet). Paraconsistent systems introduce a third value for "Both True and False" to contain the damage of a contradiction.

WVL and paraconsistent logics share a common goal: preventing a localized failure from destroying the entire logical framework. However, they use opposite strategies. A paraconsistent logic tries to "isolate" the contradiction so that the rest of the system can keep working normally. WVL "propagates" the W state precisely because it wants the system to stop. In WVL, the "explosion" of the W state is a feature, not a bug. It acts as an automated alarm system. Where a paraconsistent logic might say, "We have a contradiction here, but let's ignore it and keep flying the plane," WVL says, "We have a structural failure here; the pilot must take over immediately because the logic core is compromised."

9.4 Comparison with Programming "Null" and "NaN"

In computer science, the concepts of Null (in SQL or Java) and NaN (Not a Number in IEEE 754) are the closest functional relatives to the W state. In SQL, Null represents "missing information," and its behavior is similar to Kleene logic (e.g., True OR Null is often True). In floating-point math, NaN is an absorbing element: any operation with NaN results in NaN (e.g., $5 + \text{NaN} = \text{NaN}$).

WVL can be seen as the formalization of the "NaN" principle into the very fabric of propositional logic. However, while NaN is limited to arithmetic, W is integrated into the Boolean operators (AND, OR, NOT). Furthermore, "Null" in programming is often a pointer to "nothing," which leads to the infamous NullPointerException. In WVL, W is a first-class citizen of the logic set. It doesn't cause a "crash" in the sense of a system halt; it causes a "logical result of W." This allows the program to continue executing but within a "failure-aware" state, providing a more graceful and predictable way to handle errors than the typical "exception-handling" blocks found in modern high-level languages.

9.5 Summary Table of Operational Differences

The following table summarizes how these various systems treat their third value (represented here as '?' for general comparison) when combined with a True (+1) value:

Logic System	Operator	Input	Output	Philosophy
Classical	OR	+1, -1	+1	Bivalence
Kleene (K3)	OR	+1, ?	+1	Optimistic (Unknown could be true)
Lukasiewicz	OR	+1, ?	+1	Probabilistic/Potentiality
Fuzzy	OR (Max)	1.0, 0.5	1.0	Degree of membership
WVL	OR	+1, W	W	Strict (Integrity over truth)
SQL	OR	True, Null	True	Informational (Missing data)

This comparison clarifies that WVL is uniquely "pessimistic." It is the only system in the group that allows a non-false, non-null value to be "defeated" by the third state. This is the necessary trade-off for high-integrity systems: we sacrifice the ability to "guess" a true result in exchange for the certainty that any "True" result we do receive is 100% free of structural corruption. While K3 and L3 are better for data mining and linguistics, WVL is the superior choice for the logic cores of autonomous machines and life-support systems.

10. Case Study: Aerospace and Autonomous Vehicle Logic Cores

The theoretical superiority of Wrong-Valued Logic (WVL) is most clearly demonstrated when applied to the high-stakes environments of aerospace engineering and autonomous vehicle (AV) navigation. In these domains, the primary challenge is not simply reaching a correct decision,

but ensuring that no incorrect decision is ever mistaken for a correct one. Current systems rely heavily on redundancy and voting algorithms to mask errors. However, as systems become more complex, the risk of "common-mode" failures and the subtle corruption of voting logic increase. This section examines how replacing traditional binary or Kleene-based logic cores with WVL-integrated architectures can prevent catastrophic outcomes in real-world failure scenarios.

10.1 The Redundancy Trap in High-Altitude Flight Systems

In modern avionics, Triple Modular Redundancy (TMR) is the gold standard. Three independent flight computers perform the same calculation, and a "voter" selects the majority result. If one computer fails and outputs an incorrect value, the other two override it. This works well for "hard" failures where a computer stops responding entirely. However, it is vulnerable to "soft" failures where a computer provides a value that is mathematically valid but contextually incorrect.

Consider a flight control surface command where the valid range is -1 (Full Down) to +1 (Full Up). In a binary-based TMR system, if Computer A outputs +1, Computer B outputs +1, and Computer C suffers a bit-flip in its memory that causes it to output -1, the voter sees two +1s and moves the wing flap up. But if a more complex corruption occurs—perhaps an EMI event that affects the voltage levels of the voter itself—the binary logic may "round" a corrupted 2.5V signal to a valid +1. The system proceeds under the illusion of consensus.

By implementing WVL, the individual computers do not just output values; they output integrity states. If Computer C detects an internal parity error or an out-of-range sensor voltage, it outputs W. The WVL-voter logic follows the rule: $(+1 \text{ AND } +1 \text{ AND } W) = W$. Instead of "guessing" that the majority is right, the flight core immediately recognizes that the redundancy has been compromised. In a high-altitude cruise, this triggers an immediate transition to a "Direct Law" manual mode or a stable-state autopilot, rather than allowing a corrupted voter to potentially drive the aircraft into an unrecoverable maneuver.

10.2 Sensor Fusion and the "Ghost Obstacle" Problem in AVs

Autonomous vehicles rely on sensor fusion, combining data from LiDAR, Radar, and Cameras to build a world model. A common failure mode is the "false negative," where a camera might miss an obstacle due to glare while the Radar detects it. Traditional "Bayesian" fusion often assigns weights to these sensors. If the Camera (weighted at 0.7) says "Path Clear" (+1) and the Radar (weighted at 0.3) says "Obstacle" (-1), the weighted average might still stay above the "Brake" threshold, leading to a collision.

In a WVL-based fusion core, we treat sensor "blindness" as a W state rather than a low-probability state. If a Camera is blinded by direct sunlight, it does not output "Path Clear"; it outputs W. Following the WVL OR operator rule: $(W \text{ OR } -1) = W$. The fusion core receives a W

result, which is interpreted as "Sensor blind, environment unknown." Unlike a binary system that might "confidently" drive into the obstacle because it trusts the camera's clear-path signal, the WVL system identifies the structural failure of its perception layer. The vehicle's response is not a "best guess" but a "safe halt" (e.g., emergency braking or pulling over), as the logic itself cannot confirm the safety of the path.

10.3 Scenario A: The Single-Event Upset (SEU) in a Voting Circuit

Single-Event Upsets, often caused by cosmic rays, can flip a bit in a processor's logic unit. In a critical "Brake Enable" circuit, a bit flip might change a "0" to a "1" at the exact moment a safety check is performed. If the logic is binary, the "1" is accepted as a valid command.

In a WVL-aware hardware gate, an SEU often results in an "illegal" state (such as both rails of a dual-rail signal being high). Because W is an absorbing element, this illegal state does not just vanish at the next clock cycle. It propagates. If the "Brake Enable" check involves an AND operation with an SEU-corrupted W value, the result is W. The system's safety monitor sees the W and understands that the "Brake Enable" signal is untrustworthy. It can then trigger a secondary, purely mechanical fail-safe. WVL ensures that the "Wrongness" created by the cosmic ray is treated as a first-class logical entity that must be handled, rather than a transient noise that might be accidentally integrated into a critical command.

10.4 Scenario B: Asymmetric Communication Failures

In distributed systems, the "Byzantine Generals' Problem" describes a scenario where different components receive different information about a failure. A sensor might be failing in such a way that it sends a "Valid" signal to the engine controller but a "Failure" signal to the cockpit display. This asymmetry can lead to conflicting automated actions.

WVL mitigates this through "Strict Messaging." If the engine controller is WVL-enabled, any signal it receives that is not a definitive +1 or -1 is treated as W. If the sensor starts oscillating or providing marginal voltages, the engine controller does not attempt to "filter" the noise. It absorbs the noise as W and broadcasts its own state as W to the rest of the network. This forces a system-wide "Wrong" consensus. The logic prevents the engine from continuing to run on "marginal" data while the cockpit is trying to shut it down. By making W an absorbing, non-maskable value, we ensure that the "Byzantine" behavior of a failing component is immediately "leveled" to a uniform failure state across all nodes.

10.5 Transitioning from Fail-Safe to Fail-Wrong Architectures

The ultimate takeaway from these case studies is the shift from "Fail-Safe" (where we try to guess a safe state) to "Fail-Wrong" (where we acknowledge the loss of logic). In current architectures, a "Safe" state is often just a default binary value (e.g., 0 for "No Throttle").

However, if the failure itself is in the logic that determines the throttle position, the "0" might be just as corrupted as a "1".

A Fail-Wrong architecture using WVL doesn't default to a value; it defaults to a state of invalidity. In the aerospace and AV examples, this distinction is life-saving. It moves the "Error Handling" out of the unpredictable realm of software "Catch" blocks and into the predictable, mathematically guaranteed realm of the logic core. When the logic says W, the machine stops because it no longer "knows" how to be safe. This intellectual honesty in machine reasoning—acknowledging when the internal world model is broken—is the primary contribution of $\{-1, W, +1\}$ logic to the field of autonomous safety.

11. Software Engineering and Compiler Integration

The theoretical and hardware foundations of Wrong-Valued Logic (WVL) necessitate a corresponding evolution in software engineering practices. In traditional programming environments, error handling is often treated as an "out-of-band" process, utilizing mechanisms such as exception throwing, return codes, or null pointers. These methods decouple the error state from the data itself, creating a significant cognitive load for developers and introducing the risk of unhandled edge cases. By integrating WVL directly into the type systems and compilers of modern programming languages, we can transform error handling from a reactive necessity into a proactive, mathematically guaranteed property of the code.

11.1 The W-Primitive and Type System Evolution

The first step in software integration is the introduction of a three-valued logic primitive, which we can denote as a *wbool* (Wrong-aware Boolean). Unlike a standard Boolean that occupies one bit (or one byte for alignment), a *wbool* must be capable of representing three distinct states: $\{-1, W, +1\}$. At the compiler level, this can be implemented using two bits or by leveraging hardware-specific W-state detection as described in Section 8.

When a variable is declared as a *wbool*, the compiler enforces the propagation rules defined in the WVL algebra. Any expression involving a *wbool* becomes a "strict" expression. For instance, if a developer writes:

```
wbool result = conditionA && conditionB;
```

The compiler does not generate standard binary AND instructions. Instead, it generates a W-aware sequence that checks for the structural integrity of both *conditionA* and *conditionB*. If either is W, *result* becomes W. This shifts the responsibility of error checking from the developer to the compiler's type-inference engine. A variable of type *wbool* cannot be used in a context

that expects a standard binary result without an explicit "handler" that addresses the W state, ensuring that "Wrongness" is never accidentally ignored.

11.2 W-Propagation vs. Traditional Exception Handling

In high-level languages like C++ or Java, errors are typically managed through "try-catch" blocks and stack unwinding. While powerful, exception handling is computationally expensive and non-deterministic in its timing—making it unsuitable for hard real-time systems. Furthermore, exceptions often "jump" over large sections of logic, making it difficult to trace the exact state of the system at the moment of failure.

WVL provides an alternative: Value-Based Propagation. Instead of throwing an exception, a function that encounters a structural failure simply returns W. Because W is an absorbing element, this "Wrongness" naturally flows through the calling code.

Consider the following pseudo-code:

```
wbool sensorVal = readSensor(); wbool processed = process(sensorVal); wbool command =  
calculateCommand(processed);
```

If `readSensor()` returns W, the `processed` variable automatically becomes W, as does the `command` variable. No explicit "if (error)" checks are required between these steps. The logic is "self-handling." At the end of the chain, the system only needs to check the final command. If it is W, the system enters its fail-safe routine. This model provides the safety of exception handling with the performance and predictability of standard data flow.

11.3 Static Analysis and Dead-Path Elimination

Integrating WVL into the compiler allows for more advanced static analysis during the build process. Compilers can perform "Wrongness Analysis" to identify code paths that are mathematically guaranteed to result in W under certain hardware failure models. This is particularly useful for formal verification.

By analyzing the logic tree, a W-aware compiler can detect "Error-Prone Sinks"—sections of code where multiple independent W-sources converge. If the compiler determines that a critical safety function depends on a logical structure that is highly susceptible to W-poisoning (due to lack of redundancy or poor gate isolation), it can flag this as a structural vulnerability during compilation. Furthermore, the compiler can optimize W-logic by performing dead-path elimination: if a branch condition is statically determined to be W, the compiler can prune the entire branch and replace it with a terminal W-state, reducing the runtime overhead of checking corrupted logic.

11.4 Memory Safety and Structural Integrity

One of the most common sources of software failure is memory corruption, such as buffer overflows or dangling pointers. In a standard system, a corrupted pointer is just a value that points to the "wrong" memory address, often leading to a crash only when that address is accessed.

In a WVL-integrated memory model, a pointer is not just an address; it is a W-aware reference. If a memory management unit (MMU) detects an illegal access or a parity error in a pointer's value, it can mark that pointer as W. Any attempt to dereference a W-pointer results in a W-value being loaded into the target register. This prevents the "silent" propagation of corrupted data. Instead of the program crashing with a segmentation fault at an unpredictable time, the corruption is immediately reflected in the logic of the program. The "Wrongness" of the pointer becomes the "Wrongness" of the data, which then propagates through the WVL operators to safely halt the affected sub-system.

11.5 Syntax for Explicit W-Handling

To make WVL practical for developers, languages must provide clear syntax for resolving the W state when it reaches a system boundary. We propose the "Integrity Match" construct, a ternary version of the traditional if-else statement:

```
match(wbool x) { case +1: // Execute if True case -1: // Execute if False case W: // Execute if  
Structural Failure detected }
```

This forces the developer to acknowledge the three-state reality of the system. Unlike an "else" block, which might catch both "False" and "Error" states in a binary system, the W-case is distinct. This ensures that the recovery logic for a "False" condition (e.g., "The door is not locked, so try to lock it") is never confused with the recovery logic for a "Wrong" condition (e.g., "The door sensor is broken, so sound the alarm"). By embedding these distinctions into the language syntax, WVL-integrated software becomes inherently more robust and easier to audit for safety compliance.

12. Conclusion and Future Research Directions

The formalization of the $\{-1, W, +1\}$ logic system represents a necessary departure from the probabilistic and uncertainty-based models that have dominated non-classical logic for the past century. By replacing the neutral, often-ignored "zero" or "unknown" state with a strictly defined, absorbing "Wrong" state, we provide a mathematical framework that aligns with the physical realities of hardware failure and structural corruption. This paper has demonstrated

that a logic designed for integrity rather than just truth resolution is the essential missing component in the architecture of modern safety-critical systems.

12.1 Synthesis of the Wrong-Valued Paradigm

The core contribution of this work is the establishment of Wrong-Valued Logic (WVL) as a distinct algebraic entity. We have moved beyond the "optimistic" masking of errors found in Kleene and Lukasiewicz logics, proving that in environments where safety is paramount, an error must be treated as a terminal contagion rather than a temporary state of indeterminacy. Through the definition of W-dominant operators, we have created a "fail-fast" logic where the presence of a structural breach invalidates the entire computational chain. This ensures that no autonomous system—be it a flight controller, a medical device, or a self-driving vehicle—can ever arrive at a "True" or "False" conclusion if the very gates and sensors used to reach that conclusion are compromised.

12.2 Contributions to Hardware and Software Reliability

This research has bridged the gap between abstract logic and physical implementation. We have explored how WVL can be realized in hardware through multi-threshold voltage mapping, high-impedance states, and dual-rail signaling. These physical layers provide a tangible manifestation of the "absorbing" property, ensuring that W is not just a value in a table but a physical override that protects the system from the unpredictability of corrupted silicon. On the software side, we have shown how integrating W as a first-class primitive in compilers and type systems can eliminate the non-determinism of traditional exception handling, replacing it with a predictable, value-based propagation of integrity.

12.3 Future Research: Higher-Order W-Logics

While this paper has focused on propositional WVL, the next frontier is the development of Higher-Order Wrong-Valued Logic. Future research should investigate how W-states behave within predicate calculus and modal logic. For instance, how does a universal quantifier (For All x , $P(x)$) behave if $P(x)$ is W for a subset of the domain? We hypothesize that W will act as a "strength-reversing" operator, where even a single W instance within an infinite domain renders the entire quantification W. Furthermore, exploring the temporal aspects of W-logic—how a "Wrong" state persists or potentially resets over time—will be critical for long-duration autonomous missions where transient errors must be cleared without compromising the logic's integrity during the event itself.

12.4 Quantum Wrongness and Semantic Breach

Another compelling avenue for future study is the application of W-logic to quantum computing. In a quantum environment, decoherence is often treated as a loss of information.

We propose a model of "Quantum Wrongness" where decoherence is not just a return to a classical state, but a structural failure of the qubit's logical standing. Developing a "W-gate" for quantum circuits could provide a formal mechanism for identifying and isolating qubits that have undergone semantic breach, potentially leading to new forms of quantum error correction that prioritize state-integrity over state-recovery.

12.5 Toward a Unified Standard for Failure Logic

The ultimate goal of this research is the standardization of the W-primitive across the global engineering stack. Just as IEEE 754 standardized the handling of "NaN" in floating-point arithmetic, there is a dire need for a universal standard for "Wrongness" in discrete logic. Such a standard would allow for seamless interoperability between sensors, logic cores, and actuators from different manufacturers, ensuring that a "W" signal from a camera is interpreted with the same strictness by an engine controller.

In conclusion, the shift from $\{-1, 0, +1\}$ to $\{-1, W, +1\}$ is more than a change in notation; it is a change in philosophy. It is an admission that our machines are fallible and that our logic must account for that fallibility at its most fundamental level. By embedding the concept of "Wrongness" into the heart of our calculations, we do not make our systems weaker; we make them more resilient, more self-aware, and ultimately, more trustworthy. The calculus of failure is the foundation upon which the next generation of safe, autonomous intelligence will be built.

13. References

Foundational Logic and Multi-Valued Systems

- Belnap, N. D. (1977). A useful four-valued logic. *Modern Uses of Multiple-Valued Logic*, 5-37.
- Birkhoff, G. (1948). *Lattice Theory*. American Mathematical Society Colloquium Publications, Vol. 25.
- Ginsberg, M. L. (1988). Multivalued logics: A uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4(3), 265-316.
- Kleene, S. C. (1952). *Introduction to Metamathematics*. North-Holland Publishing Company.
- Lukasiewicz, J. (1920). On three-valued logic. *Ruch Filozoficzny*, 5, 170-171.
- Malinowski, G. (1993). *Many-Valued Logics*. Oxford University Press.
- Post, E. L. (1921). Introduction to a general theory of elementary propositions. *American Journal of Mathematics*, 43(3), 163-185.
- Priest, G. (2008). *An Introduction to Non-Classical Logic: From If to Is*. Cambridge University Press.
- Rescher, N. (1969). *Many-Valued Logic*. McGraw-Hill.

Fault-Tolerant Computing and Hardware Design

- Avizienis, A. (1967). Design of fault-tolerant computers. *AFIPS '67 (Fall)*, 733-743.
- Eichelberger, E. B., & Williams, T. W. (1977). A logic design structure for LSI testability. *Proceedings of the 14th Design Automation Conference*, 462-468.
- Johnson, B. W. (1989). *Design and Analysis of Fault-Tolerant Digital Systems*. Addison-Wesley.
- Kopetz, H. (2011). *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Science & Business Media.
- Pradhan, D. K. (1996). *Fault-Tolerant Computer System Design*. Prentice-Hall, Inc.
- Shooman, M. L. (2002). *Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*. John Wiley & Sons.

- Siewiorek, D. P., & Swarz, R. S. (1998). *Reliable Computer Systems: Design and Evaluation*. AK Peters/CRC Press.

Safety-Critical Systems and Formal Methods

- Bowen, J. P., & Hinchey, M. G. (1995). Ten commandments of formal methods. *Computer*, 28(4), 56-63.
- Knight, J. C. (2012). *Fundamentals of Dependable Computing for Software Engineers*. CRC Press.
- Leveson, N. (1995). *Safeware: System Safety and Computers*. Addison-Wesley.
- Leveson, N. (2011). *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press.
- Rushby, J. (1993). *Formal Methods and the Certification of Critical Systems*. SRI International.
- Storey, N. (1996). *Safety-Critical Computer Systems*. Addison-Wesley.

Programming Languages and Software Engineering

- Hoare, C. A. R. (2009). Null References: The Billion Dollar Mistake. *QCon London*.
- IEEE Computer Society. (2008). *IEEE Standard for Floating-Point Arithmetic (IEEE 754-2008)*.
- Meyer, B. (1992). Applying "design by contract". *Computer*, 25(10), 40-51.
- Pierce, B. C. (2002). *Types and Programming Languages*. MIT Press.
- Wadler, P. (1992). The essence of functional programming. *Proceedings of the 19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1-14.

Industry Standards and Guidelines

- DO-178C. (2011). *Software Considerations in Airborne Systems and Equipment Certification*. RTCA.
- IEC 61508. (2010). *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems*. International Electrotechnical Commission.
- ISO 26262. (2018). *Road vehicles – Functional safety*. International Organization for Standardization.

- MISRA C:2012. (2013). *Guidelines for the use of the C language in critical systems*. Motor Industry Software Reliability Association.

Digital Resources and Technical Reports

- Association for Computing Machinery (ACM) Digital Library: [<https://dl.acm.org>]
- IEEE Xplore Digital Library: [<https://ieeexplore.ieee.org>]
- NASA Formal Methods Technical Reports: [<https://shemesh.larc.nasa.gov/fm/>]
- National Institute of Standards and Technology (NIST) Publications: [<https://www.nist.gov/publications>]

The author has published over 5,300 AI-assisted papers at:

<https://www.researchgate.net/profile/Douglas-Youvan/research> . Google Scholar has indexed these preprints, and if you want a particular reference, the AI mode of a Google search (Gemini) has efficiently catalogued these preprints.

